Managing Your Analysis Debt

by Mary Gorman and Ellen Gottesdiener

We recently heard about an agile project that implemented four user stories. The first story began, "As a sales associate ..."; The second story began, "As a sales rep ..."; and the third and fourth stories began, "As a sales consultant ..." These stories and the resulting software successfully delivered the customer's understanding.

Now, flash forward three iterations. The customer has had an epiphany: These three users are really only one—the sales rep. So the stories, rules, data, and interface experiences are overlapping and conflicting, and they need to be refactored. This change cost the team almost two iterations of work.

ISTOCKPHOTO

Technical Debt in Software Projects

Ward Cunningham coined the metaphor of technical debt in 1992. "Shipping first-time code is like going into debt," he said. "A little debt speeds development so long as it is paid back promptly with a rewrite. The danger occurs when the debt is not repaid." [1]

For large software projects, using debt is often a wise financial strategy. But incurring debt is always a risk, especially if it is high-interest debt and you're not paying close attention to the cost. The same is true of technical debt, and it applies not only to code but also to architectural design [2] and even to requirements analysis.

Analysis Debt

Analysis debt results when a team:

- Defines and implements requirements in a manner that limits or disables future integration of various customer views, extensibility, scalability, or reuse
- Uses requirements practices that make it harder to make changes later, require extra effort to clean up, and generally cost more
- Invests too much in analysis too early

Analysis debt can be intentional or unintentional. For example, customers may *intentionally* limit requirements for the next release, which can lead to rework in subsequent releases to "fix" the resulting product. On the other hand, a domain subject matter expert may *unintentionally* provide inadequate product details or make poor choices of what requirements to deliver.

No development method is immune to analysis debt. On traditional (waterfall) projects, teams can overinvest in analysis of unnecessary requirements. Later, the team finds itself reprioritizing and removing requirements.

On agile projects, teams are addicted to delivery (in a good way!). Producing working, tested software of value as soon as possible pays off for the business. Yet, it can have the unintended consequence of incurring analysis debt.

Good Debts and Bad

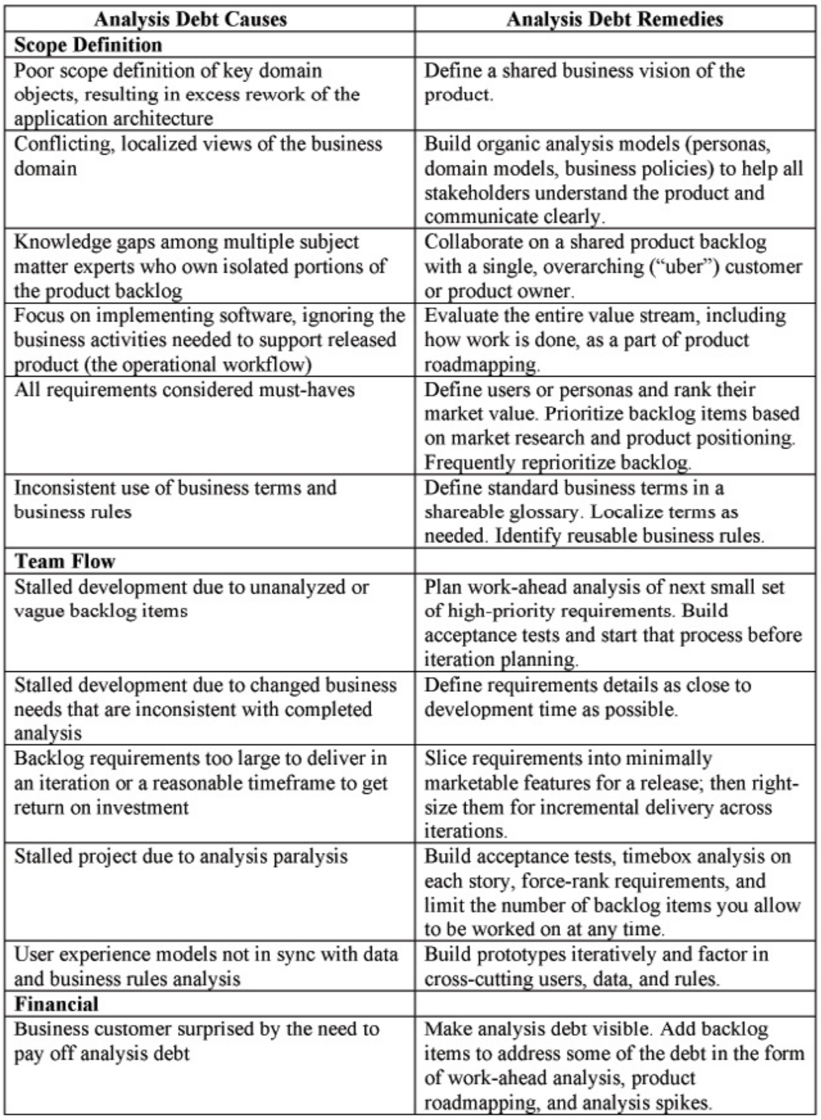
Like cholesterol, analysis debt comes in good and bad versions. Some examples of good analysis debt are:

- Delaying detailed requirements analysis of low-priority requirements
- Delaying scheduling and analyzing volatile requirements, saving wasted time and effort in the face of uncertainty
- Pretending that user requirements dependencies don't exist, so that you deliver business-crucial features while allowing the delivery team time to learn how to deliver efficiently

Recognizing the causes of bad analysis debt can help you prevent or, at least, actively manage it. Table 1 shows common forms of analysis debt we have observed, along with suggested remedies.

Responsible Analysis Debt

The stakeholders of analysis debt—the business customer, the delivery team, and the senior managers who advise the customer and team—must collaboratively decide when, why, and how to incur analysis debt. Everyone should be



Analysis Debt Causes	Analysis Debt Remedies
Scope Definition	
Poor scope definition of key domain objects, resulting in excess rework of the application architecture	Define a shared business vision of the product.
Conflicting, localized views of the business domain	Build organic analysis models (personas, domain models, business policies) to help all stakeholders understand the product and communicate clearly.
Knowledge gaps among multiple subject matter experts who own isolated portions of the product backlog	Collaborate on a shared product backlog with a single, overarching ("uber") customer or product owner.
Focus on implementing software, ignoring the business activities needed to support released product (the operational workflow)	Evaluate the entire value stream, including how work is done, as a part of product roadmapping.
All requirements considered must-haves	Define users or personas and rank their market value. Prioritize backlog items based on market research and product positioning. Frequently reprioritize backlog.
Inconsistent use of business terms and business rules	Define standard business terms in a shareable glossary. Localize terms as needed. Identify reusable business rules.
Team Flow	
Stalled development due to unanalyzed or vague backlog items	Plan work-ahead analysis of next small set of high-priority requirements. Build acceptance tests and start that process before iteration planning.
Stalled development due to changed business needs that are inconsistent with completed analysis	Define requirements details as close to development time as possible.
Backlog requirements too large to deliver in an iteration or a reasonable timeframe to get return on investment	Slice requirements into minimally marketable features for a release; then right-size them for incremental delivery across iterations.
Stalled project due to analysis paralysis	Build acceptance tests, timebox analysis on each story, force-rank requirements, and limit the number of backlog items you allow to be worked on at any time.
User experience models not in sync with data and business rules analysis	Build prototypes iteratively and factor in cross-cutting users, data, and rules.
Financial	
Business customer surprised by the need to pay off analysis debt	Make analysis debt visible. Add backlog items to address some of the debt in the form of work-ahead analysis, product roadmapping, and analysis spikes.

Table 1: Analysis debt causes and remedies

cognizant of the factors that justify the cost of analysis debt for your product and market—for example, earning short-term revenue, delivering differentiating features to a swiftly moving market, or being first to market.

In some cases, it may be prudent to incur the cost of analysis debt if the realized income outweighs the risks and cost of future nimbleness. Be sure to incorporate the expected life of the product into your analysis-debt assessment. Most important, be sure that the consequences of deferring the cost of repaying analysis debt (rework, repair, and waste) are transparent to everyone.

The final accounting is clear: Analysis debt will compound the longer you go without paying it off. Be a prudent investor! Implement strategies to protect against unintentional analysis debt, and carefully and actively manage your intentionally incurred analysis debt. {end}

How do you detect and mitigate analysis debt on your projects?

Follow the link on the StickyMinds.com homepage to join the conversation.

Sticky Notes

For more on the following topics go to www.StickyMinds.com/bettersoftware.

- References
- Recommended links

Let your voice be heard!

Take the *Better Software* magazine reader survey today.

WWW.SQE.COM/GO?BSM09SURVEY

BETTER SOFTWARE MAGAZINE